Exercise 1

Write an algorithm that copies all keys that are stored in a binary search tree into an array of appropriate size. In the resulting array, the keys shall be sorted in descending order.

Solution:

The main idea of the algorithm is to rely on recursion and:

1. copy all keys of the right subtree to the array (in sorted order, using the algorithm recursively);
2. copy the root of the tree into the array;
3. copy all keys of the left subtree to the array (in sorted order, using the algorithm recursively).

Using pseudo code, the algorithm can be written as:

```plaintext
Tree2Array(T: BinTree, A: Array[1..n], pos:Integer) : Integer {
   /* write elements of the binary search tree T into array A,
      starting at position pos;
      return index of next empty element in A
   */
   if (T != emptyTree) { /* if T is not an empty tree */
      pos := Tree2Array(T.rightSon,A,pos);
      A[pos] := T.key; pos := pos + 1;
      pos := Tree2Array(T.leftSon,A,pos);
   }
   return pos;
}
```

Exercise 2

Consider the binary tree given by the expression
x = (5, (3, emptyTree, (4, emptyTree, emptyTree)),
     (8, (6, emptyTree, emptyTree), (10, (9, emptyTree, emptyTree),
      (13, emptyTree, emptyTree))))

- draw a diagram of this binary tree and decide whether its a binary search tree

![Binary Tree Diagram]

For each node, all keys in the left subtree are smaller than that in the node, and all keys in the right subtree are larger. Hence, the tree is a binary search tree.

- perform the following operations (using the resp. algorithms from the lectures), and draw a diagram of the search tree after each operation:
  - TREE_INSERT(x, 11)
    ![Tree After Insert 11]
  - TREE_DELETE(x, 5)
    ![Tree After Delete 5]
  - TREE_INSERT(x, 5)
    ![Tree After Insert 5]
  - TREE_INSERT(x, 12)
    ![Tree After Insert 12]
Exercise 3

Decide whether the binary tree given in exercise IV is an AVL tree

• before the insert/delete operations, and

• after each of the regular insert/delete operations.

Again, perform the insert/delete operations given in exercise IV, and name and perform the rotation(s) to restore the AVL property after each step (if required). Draw a diagram of the search tree after each of your insert/delete, or rotation operations.

Solution:

Before the insert/delete operations, the height balances for the nodes are:

```
   6
  / \
 3   8
 /   \
4   10
 /     \\
5     9
       /\
       11
     /   \
    12
```

Therefore, the binary search tree is also an AVL tree.

• after TREE_INSERT(x,11), the AVL property is violated in node 8 → left-rotation on node 8:

```
   6
  / \   
 3   8
 /   / \
4   10
 /     \\
5     11
       /\
       12
```

• after TREE_DELETE(x,5), we still have an AVL tree → rotation required:

```
   6
  /  \
 3   13
 /    \\n4     11
```

• after TREE_INSERT(x,5), the AVL property is violated in node 3, which requires a left-rotation on node 3:
• after TREE_INSERT(x, 12), the AVL property is violated in node 13, and a left-right-rotation is required: